

리버스 엔지니어링

- 실행파일(기계어)만 있는 상태에서 코드의 내용을 추론하는 과정
 - 기계어는 사람이 읽을 수 없음
 - 어셈블리어로 변환(disassemble)**하여 분석해야 함
 - 리버싱을 하려면 반드시 어셈블리어를 읽고 해석할 수 있어야 함
-
- 소스를 역추적하는 것을 말함.
 - 소스코드를 빌드해서 만들어진 exe, dll의 바이너리를 분석해 원래의 소스코드가 어떤 식으로 만들어져 있는지 파악한다.



Microsoft Visual Studio



visual studio community



전체 쇼핑 이미지 동영상 짧은 동영상 뉴스 웹 : 더보기



Visual Studio

[https://visualstudio.microsoft.com › community](https://visualstudio.microsoft.com/community)

Visual Studio Community | Download Latest Free Version

2025. 4. 30. — Visual Studio Community · Visual Studio Community 2022. Free, fully-featured IDE for students, open-source and individual developers. · Visual ...

License Terms

Sign in to your account Follow us Visual Studio on Twitter Visual ...

Visual Studio 2013 Releases

Visual Studio Community 2013 Release Notes released on ...

[microsoft.com](#) 검색결과 더보기 »



Visual Studio

[https://visualstudio.microsoft.com › downloads](https://visualstudio.microsoft.com/downloads)

Download Visual Studio Tools - Install Free for Windows, Mac ...

2025. 4. 23. — Download Visual Studio IDE or VS Code for free. Try out Visual Studio Pro and Enterprise editions on Windows, Mac.

Visual Studio Community

Android, iOS 및 Windows용 최신 애플리케이션뿐 아니라 웹 애플리케이션 및 클라우드 서비스를 만들기 위한 모든 기능을 갖춘 확장 가능한 무료 IDE입니다.

다운로드

Visual Studio Installer

시작하기 전에 설치를 구성할 수 있도록 몇 가지 항목을 설정해야 합니다.

개인정보처리방침에 대해 자세히 알아보려면 [Microsoft 개인정보처리방침](#)을 참조하세요.

계속하면 [Microsoft 소프트웨어 사용 조건](#)에 동의하게 됩니다.

계속(O)

Visual Studio Installer

Visual Studio 설치 관리자를 가져옵니다.

확인 중

설치하는 중





취소(C)

설치 — Visual Studio Community 2022 — 17.13.6






워크로드 개별 구성 요소 언어 팩 설치 위치

설치할 항목을 선택하는 데 도움이 필요하신가요? [추가 정보](#)

웹 및 클라우드 (4)

-  **ASP.NET 및 웹 개발**
Docker 지원이 포함된 ASP.NET Core, ASP.NET, HTML/JavaScript 컨테이너를 사용하여 웹 애플리케이션을 빌드...
-  **Azure 개발**
.NET 및 .NET Framework를 사용하여 클라우드 앱을 개발하고 리소스를 만들기 위한 Azure SDK, 도구 및 프로젝트...
-  **Python 개발**
Python에 대한 편집, 디버깅, 대화형 개발 및 소스 제어가 가능합니다.
-  **Node.js 개발**
비동기 이벤트 구동 JavaScript 런타임인 Node.js를 사용하여 확장 가능한 네트워크 애플리케이션을 빌드합니다.

데스크톱 및 모바일 (5)

-  **.NET Multi-Platform App UI 개발**
.NET MAUI와 함께 C#을 사용하여 단일 코드베이스에서 Android, iOS, Windows 및 Mac용 앱을 빌드합니다.
-  **.NET 데스크톱 개발**
.NET 및 .NET Framework와 함께 C#, Visual Basic 및 F#를 사용하여 WPF, Windows Forms 및 콘솔 애플리케이션을...
-  **C++를 사용한 데스크톱 개발**
MSVC, Clang, CMake 또는 MSBuild 등 선택한 도구를 사용하여 Windows용 최신 C++ 앱을 빌드합니다.
-  **WinUI 애플리케이션 개발**
C# 또는 선택 사항으로 C++와 함께 WinUI를 사용하여 Windows 플랫폼용 애플리케이션을 빌드합니다.
-  **C++를 사용한 모바일 개발**
C++를 사용하여 iOS, Android 또는 Windows용 플랫폼 간 애플리케이션을 빌드합니다.



C++를 사용한 데스크톱 개발 ☒
MSVC, Clang, CMake 또는 MSBuild 등 선택한 도구를 사용하여 Windows용 최신 C++ 앱을 빌드합니다.

시작



리포지토리 복제(C)

GitHub 또는 Azure DevOps 같은 온라인 리포지토리에서 코드를 가져옵니다.



프로젝트 또는 솔루션 열기(P)

로컬 Visual Studio 프로젝트 또는 .sln 파일을 엽니다.



로컬 폴더 열기(F)

폴더 내에서 코드를 탐색 및 편집합니다.



새 프로젝트 만들기(N)

시작하려면 코드 스캐폴딩과 함께 프로젝트 템플릿을 선택합니다.

코드를 사용하지 않고 계속(W) →

새 프로젝트 만들기

최근 프로젝트 템플릿(R)

최근에 액세스한 템플릿 목록이 여기에 표시됩니다.

검색창

모든 언어(L)

모든 플랫폼(P)

모든 프로젝트 형식(T)



빈 프로젝트

Windows용 C++를 사용하여 처음부터 시작합니다. 시작 파일을 제공하지 않습니다.

C++

Windows

콘솔



콘솔 앱

Windows 터미널에서 코드를 실행합니다. 기본적으로 "Hello World"를 출력합니다.

C++

Windows

콘솔



CMake 프로젝트

.sln 또는 .vcxproj 파일에 종속되지 않은 최신 플랫폼 간 C++ 앱을 빌드하세요.

C++

Windows

Linux

콘솔



Windows 데스크톱 마법사

마법사를 사용하여 고유한 Windows 앱을 만드세요.

C++

Windows

데스크톱

콘솔

라이브러리



Windows 데스크톱 애플리케이션

Windows에서 실행되는 그래픽 사용자 인터페이스를 사용하는 애플리케이션용 프로젝트입니다.

C++

Windows

데스크톱



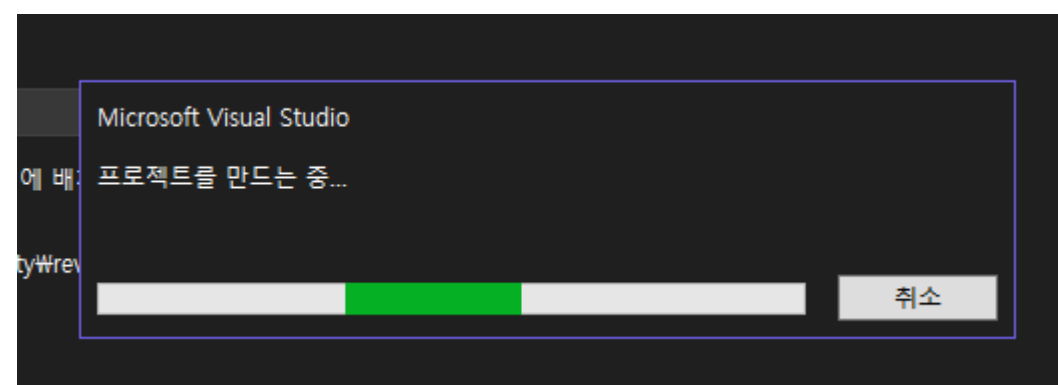
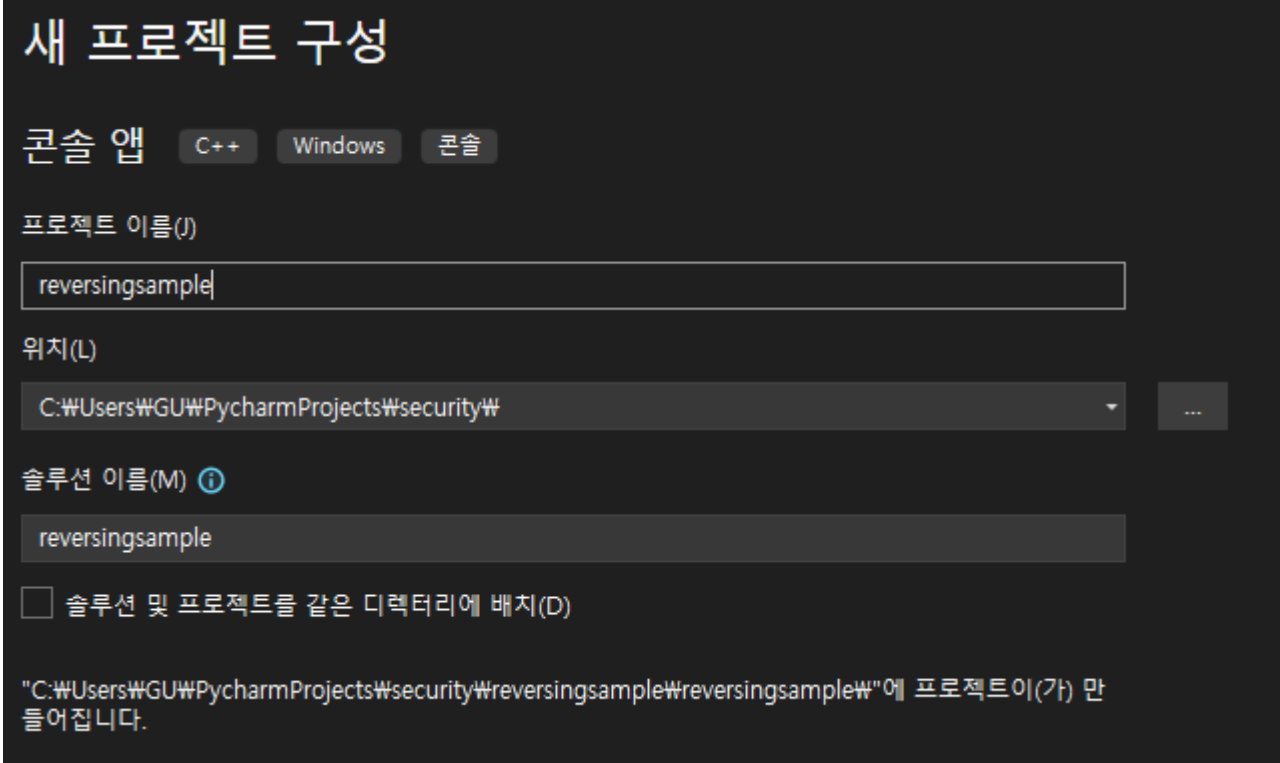
Blank TypeScript Project

A blank TypeScript Project that can be used as a console application (using node) or a library.

뒤로(B)

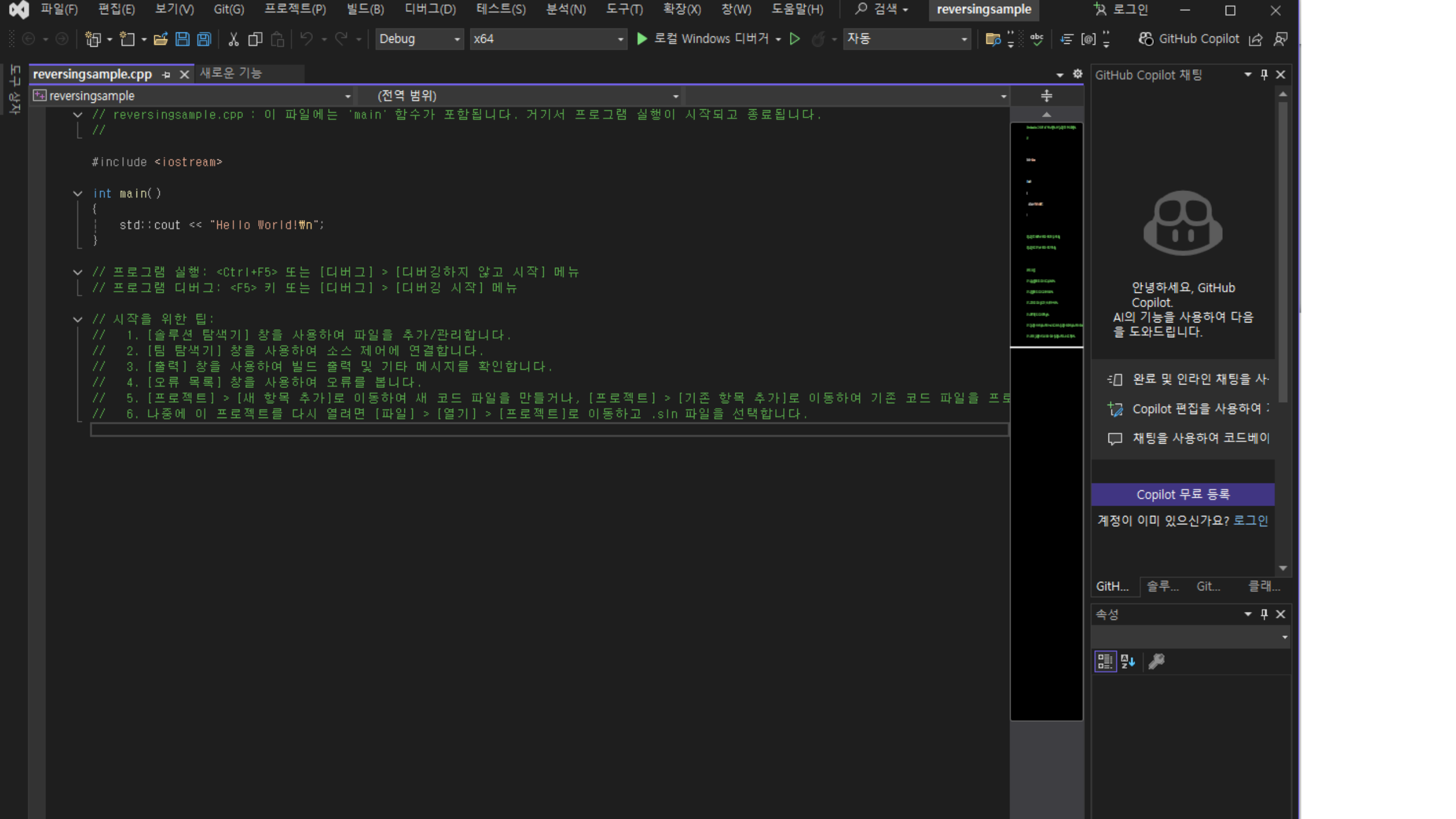
다음(N)

새 프로젝트 만들기 -> c++ 콘솔 앱



프로젝트 이름 : reversingsample

위치 : 파이참 security 경로 지정

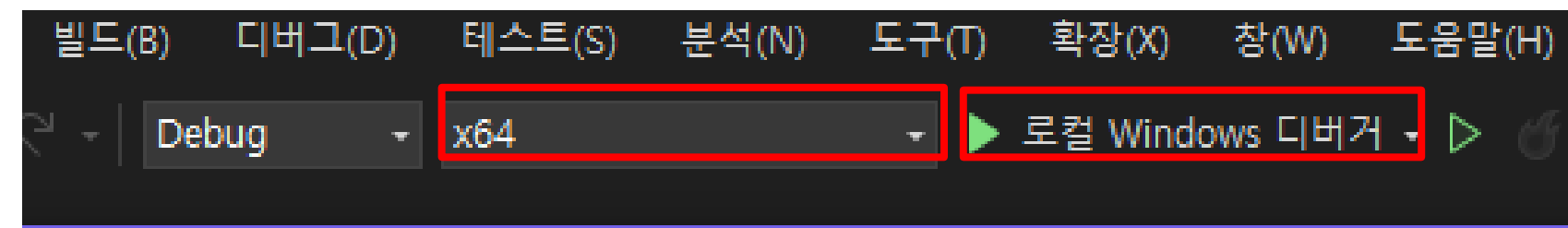
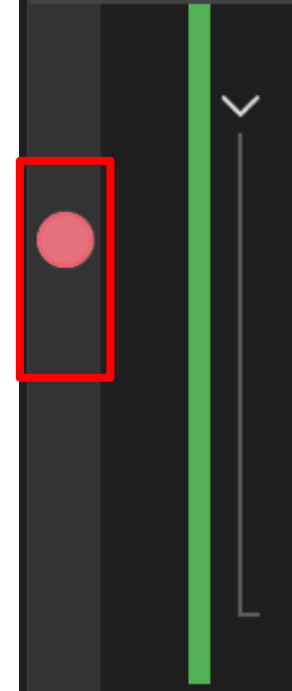
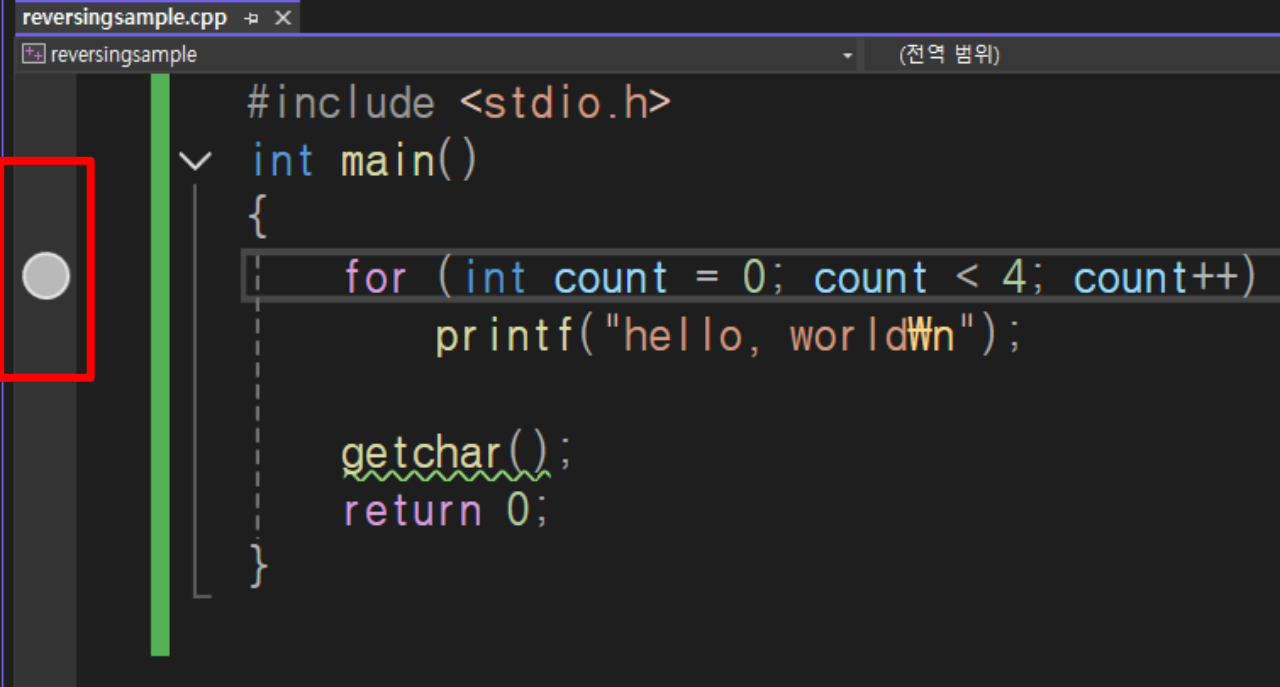


```
#include <stdio.h>
int main()
{
    for (int count = 0; count < 4; count++)
        printf("hello, world\n");

    getchar();
    return 0;
}
```


정적인 어셈블리 코드 보기

기계어 (Machine Code)	어셈블리어 (Assembly Language)	고급 언어 (C, Java, Python 등)
10111000 00000001	MOV AL, 1	int a = 1;



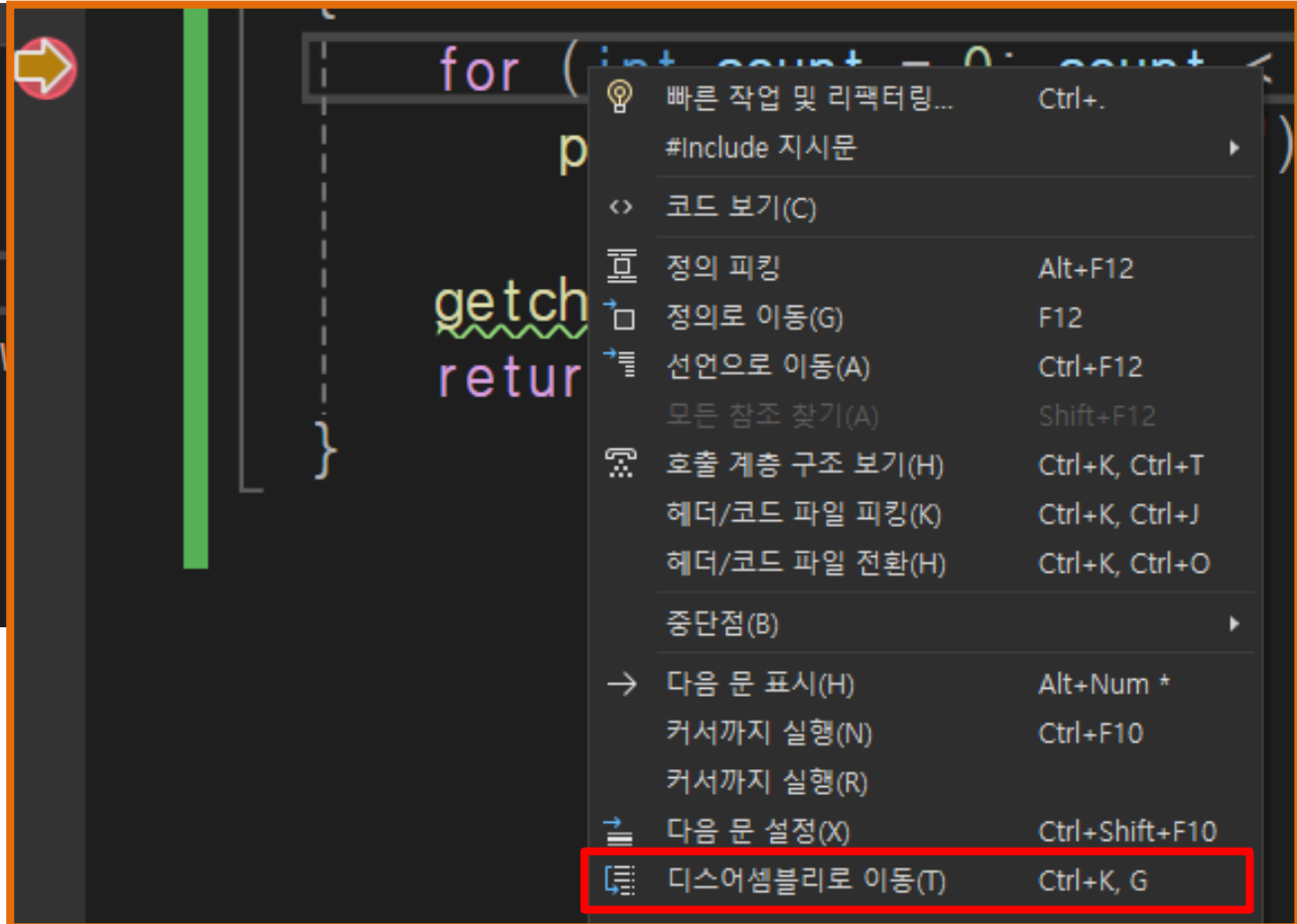
왼쪽 마우스 클릭 -> Debug : x64 선택 -> 로컬 windows 디버거 버튼 실행

reversingsample

```
#include <stdio.h>
int main()
{
    for (int count = 0;
        printf("hello, v

    getchar();
    return 0;
}
```

로컬 windows 디버거 실행



For문에서 마우스 오른쪽 클릭 -> 디스어셈블리로 이동

00007FF7F3D818AB nop

for (int count = 0; count < 4; count++)

00007FF7F3D818AC mov dword ptr [rbp+4],0
00007FF7F3D818B3 jmp main+2Dh (07FF7F3D818BDh)
00007FF7F3D818B5 mov eax,dword ptr [rbp+4]
00007FF7F3D818B8 inc eax
00007FF7F3D818BA mov dword ptr [rbp+4],eax
00007FF7F3D818BD cmp dword ptr [rbp+4],4
00007FF7F3D818C1 jge main+42h (07FF7F3D818D2h)

printf("hello, world\n");

00007FF7F3D818C3 lea rcx,[string "hello, world\n" (07FF7F3D8AC28h)]
00007FF7F3D818CA call printf (07FF7F3D81195h)
00007FF7F3D818CF nop
00007FF7F3D818D0 jmp main+25h (07FF7F3D818B5h)

getchar();

00007FF7F3D818D2 call qword ptr [__imp_getchar (07FF7F3D91308h)]
00007FF7F3D818D8 nop

return 0;

00007FF7F3D818D9 xor eax,eax
}

00007FF7F3D818DB lea rsp,[rbp+0E8h]
00007FF7F3D818E2 pop rdi
00007FF7F3D818E3 pop rbp
00007FF7F3D818E4 ret

어셈블리 코드

메모리 주소	명령어	인자(데이터 값)
for (int count = 0; count < 4; count++)		
00007FF7F3D818AC	mov	dword ptr [rbp+4],0
00007FF7F3D818B3	jmp	main+2Dh (07FF7F3D818BDh)
00007FF7F3D818B5	mov	eax,dword ptr [rbp+4]
00007FF7F3D818B8	inc	eax
00007FF7F3D818BA	mov	dword ptr [rbp+4],eax
00007FF7F3D818BD	cmp	dword ptr [rbp+4],4
00007FF7F3D818C1	jge	main+42h (07FF7F3D818D2h)

메모리주소

정의 : 명령이나 데이터가 저장된 메모리 위치 번호

컴퓨터가 명령을 실행하거나 데이터를 찾기 위해 사용하는 주소

0040: MOV AX, 5 0040 → 이 명령어가 저장된 메모리 주소

명령어

CPU에게 무엇을 하라고 지시하는 작업 명령

계산, 이동, 점프 등 다양한 기능 수행

MOV AX, 5 MOV → 데이터 이동

ADD AX, 3 ADD → 덧셈

JMP 0040 JMP → 주소로 점프

인자

명령어가 작동할 대상 (데이터, 레지스터, 메모리 주소 등)

명령어 옆에 붙어 동작의 입력과 출력 대상을 지정함

MOV AX, 5 ; AX는 목적지, 5는 원본 (둘 다 인자)

ADD AX, BX ; AX와 BX는 레지스터 인자

MOV AX, [1000h]; 1000h는 메모리 주소 인자

레지스터 vs 메모리

구분	레지스터	일반 메모리(RAM)
위치	CPU 내부	CPU 외부
접근 속도	매우 빠름 (나노초 수준)	느림 (마이크로초 수준)
용량	매우 작음 (수십 ~ 수백 바이트)	큼 (GB 단위)
용도	계산 중 임시 데이터 저장	프로그램, 데이터 저장
예시	AX, BX, R0, R1 등	0x1000, 0x2000 같은 주소 사용

리버스 추론과정 예제

고급 언어 (C 코드)	컴파일된 기계어	어셈블리어 (Disassembled Code)
<pre>int a = 3; int b = 5; int c = a + b;</pre>	<pre>B8 03 00 00 00 BB 05 00 00 00 01 D8</pre>	<pre>MOV EAX, 3 ; EAX 레지스터에 3 저장 MOV EBX, 5 ; EBX 레지스터에 5 저장 ADD EAX, EBX ; EAX = EAX + EBX</pre>

추론: $EAX = 3, EBX = 5 \rightarrow EAX = 3 + 5 \rightarrow 8$

리버스 필요할까?

상황	설명
고급 언어 코드 없음	배포된 .exe, .dll, .so, .apk 등에는 기계어만 포함되어 있음
컴파일된 실행 파일만 있음	기계어 → 어셈블리어로만 변환 가능, 원래의 C/Python 코드를 정확히 복원 불가
리버싱	실행 파일을 분석하여 원래 동작을 유추하는 과정
소스코드 보안	기업은 코드를 숨기기 위해 바이너리만 배포함 (소스 없이 분석해야 함)

레지스터 종류

이름	의미	비유	비트크기	사용용도
RAX/EAX	Accumulator	계산 결과 담는 주머니	64/32비트	함수의 return값, 계산 결과
RBX/EBX	Base	작업중 데이터를 담아두는 바구니	64/32비트	임시 데이터 저장
RCX/ECX	Counter	회수를 세는 손가락	64/32비트	반복문count, 문자열 처리
RDX/EDX	Data	보조 계산용 그릇	64/32비트	곱셈, 나눗셈 결과 보조
RSI/ESI	Source Index	출발지 주소	64/32비트	배열 복사 시 출발점
RDI/EDI	Destination Index	도착지 주소	64/32비트	배열 복사 시 도착점
RSP	Stack Pointer	스택 맨 위를 가르키는 손가락	64비트	함수 호출 시 스택 추적
RBP	Base Pointer	함수 기준점(바닥 표시)	64비트	함수 내부 변수 추적
RIP	Instruction Pointer	다음 명령어를 가리키는 눈	64비트	지금 실행 중인 명령 위치

명령어 주요 종류

명령어	의미	설명
mov	Move	값을 복사해서 옮긴다 ($a = b$)
add	Add	더한다 ($a = a + b$)
sub	Subtract	뺀다 ($a = a - b$)
mul	Multiply	곱한다 ($a = a * b$)
div	Divide	나눈다 ($a = a / b$)
xor	Exclusive OR	두 값이 다르면 1, 같으면 0 (초기화에도 많이 씀)
and	AND	비트끼리 AND 연산 (필터링, 마스크 용도)
or	OR	비트끼리 OR 연산 (값 합치기 용도)
cmp	Compare	두 값을 비교한다 (조건문 만들기 전 단계)
jmp	Jump	무조건 점프 (if문 없이 이동)
je	Jump if Equal	비교 결과가 같으면 점프 (if ($a == b$))
jne	Jump if Not Equal	비교 결과가 다르면 점프 (if ($a != b$))
jg	Jump if Greater	더 크면 점프 (if ($a > b$))

명령어 주요 종류

명령어	의미	설명
j1	Jump if Less	더 작으면 점프 (if (a < b))
call	Call function	함수를 호출한다
ret	Return	함수에서 돌아간다
push	Push to stack	값을 스택에 쌓는다
pop	Pop from stack	스택에서 값을 꺼낸다
nop	No Operation	아무것도 안 함 (시간 채우기용, 패딩)

자주 사용하는 명령어

명령어	설명
mov	값을 복사해서 옮긴다 ($a = b$)
add/sub	더한다 ($a = a + b$) 뺀다 ($a = a - b$)
cmp	비교 - 조건 분기 전 준비 두 값을 비교한다 (조건문 만들기 전 단계)
call,ret	함수 호출/복귀
push,pop	나눈다 ($a = a / b$)
xor eax, eax	레지스터 초기화(0으로 만들기)
lea	주소 계산(배열, 포인터 접근)
inc , dec	1증가, 감소
cmp	두 값을 비교한다 (조건문 만들기 전 단계)
jmp	무조건 점프 (if문 없이 이동)
je	비교 결과가 같으면 점프 (if ($a == b$))
jne	비교 결과가 다르면 점프 (if ($a != b$))
nop	아무것도 안 함 (시간 채우기용, 패딩)

디스어셈블리



x64dbg

기계어(0과 1)로 된 실행파일(EXE)을 사람이 읽을 수 있는 어셈블리어로 바꾸는 과정

디스어셈블리 툴

툴 이름	플랫폼	특징 및 추천 이유
x64dbg	Windows	<ul style="list-style-type: none">- 가장 많이 사용되는 무료 디버거/디스어셈블러- 실시간 실행 추적 가능- C 코드 → 어셈블리 비교에 최적
IDA Free / IDA Pro	Windows, Linux	<ul style="list-style-type: none">- 정적 분석의 끝판왕 - 함수 분석 자동화 - 유료버전은 업계 표준
Ghidra (NSA 제작)	Windows, Linux, macOS	<ul style="list-style-type: none">- 무료이지만 매우 강력 - C 수준의 의사코드(Pseudo code)도 제공- 플러그인 다양
Radare2 / Cutter	Windows, Linux	<ul style="list-style-type: none">- 터미널 기반 분석툴 (Radare2) + GUI 버전 (Cutter)- 완전 무료, 가볍고 빠름
objdump	Linux	<ul style="list-style-type: none">- GCC에서 바로 사용 가능 - 콘솔 기반으로 정적 디스어셈블 제공- 교육용으로 좋음



x64dbg

전체

이미지

동영상

쇼핑

뉴스

짧은 동영상

웹

⋮ 더보기



x64dbg

https://x64dbg.com ⋮

x64dbg

x64dbg can debug both x64 and x32 applications. There is only one interface. Built on open-source libraries.

[X64dbg's documentation!](#)

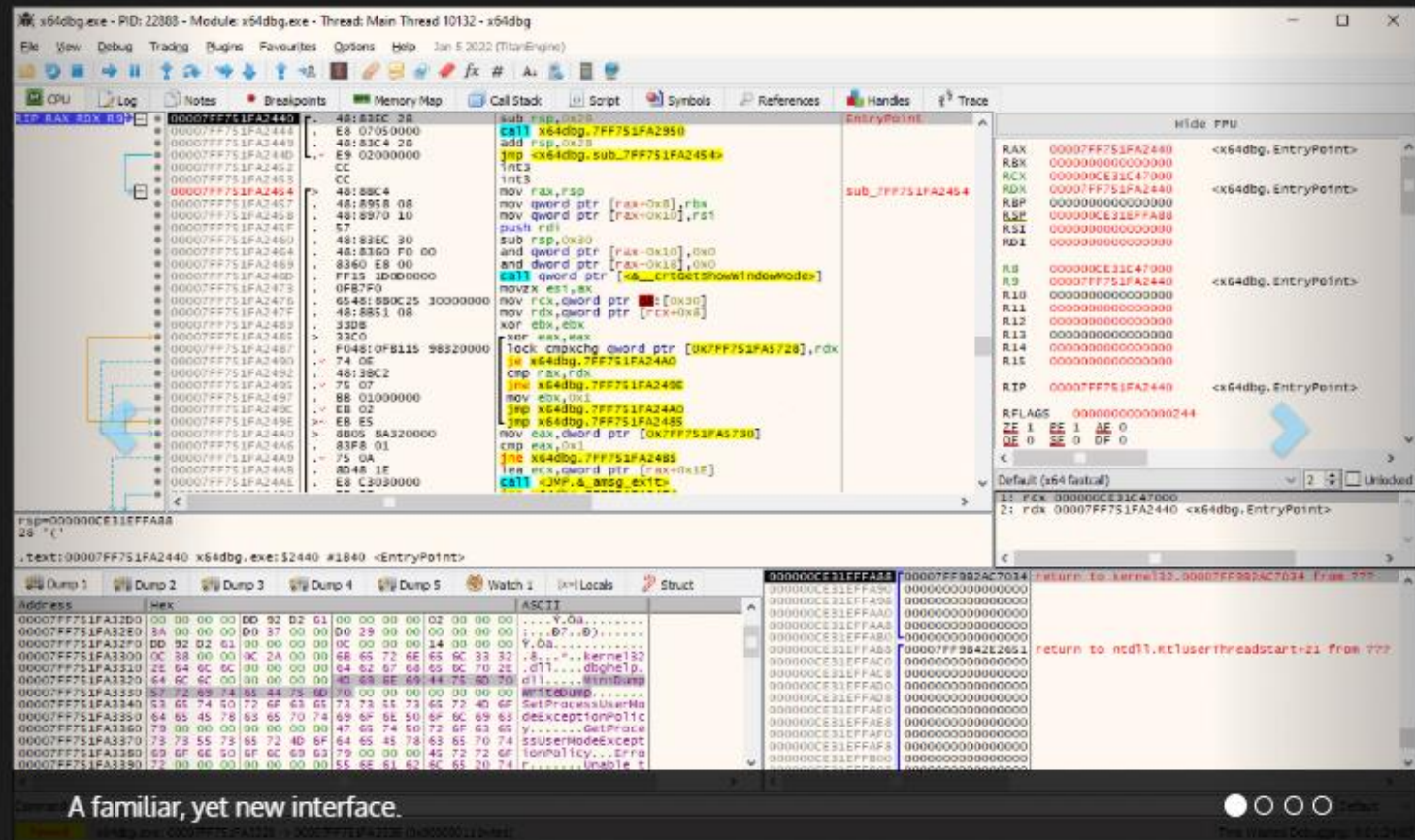
Introduction - GUI manual - Commands - ...



[Official x64dbg blog!](#)

The easiest way to do so is selecting the last instruction of ...







x64dbg Files

An open-source x64/x32 debugger for windows.

Brought to you by: [mrexodia](#)

Summary

Files

Reviews

Support

Source



Download Latest Version

snapshot_2025-03-15_15-57.zip (35.0 MB)



Get an email when there's a new version of x64dbg

Enter your email address

Next



[Home](#) / [snapshots](#)

Name	Modified	Size	Downloads / Week
------	----------	------	------------------

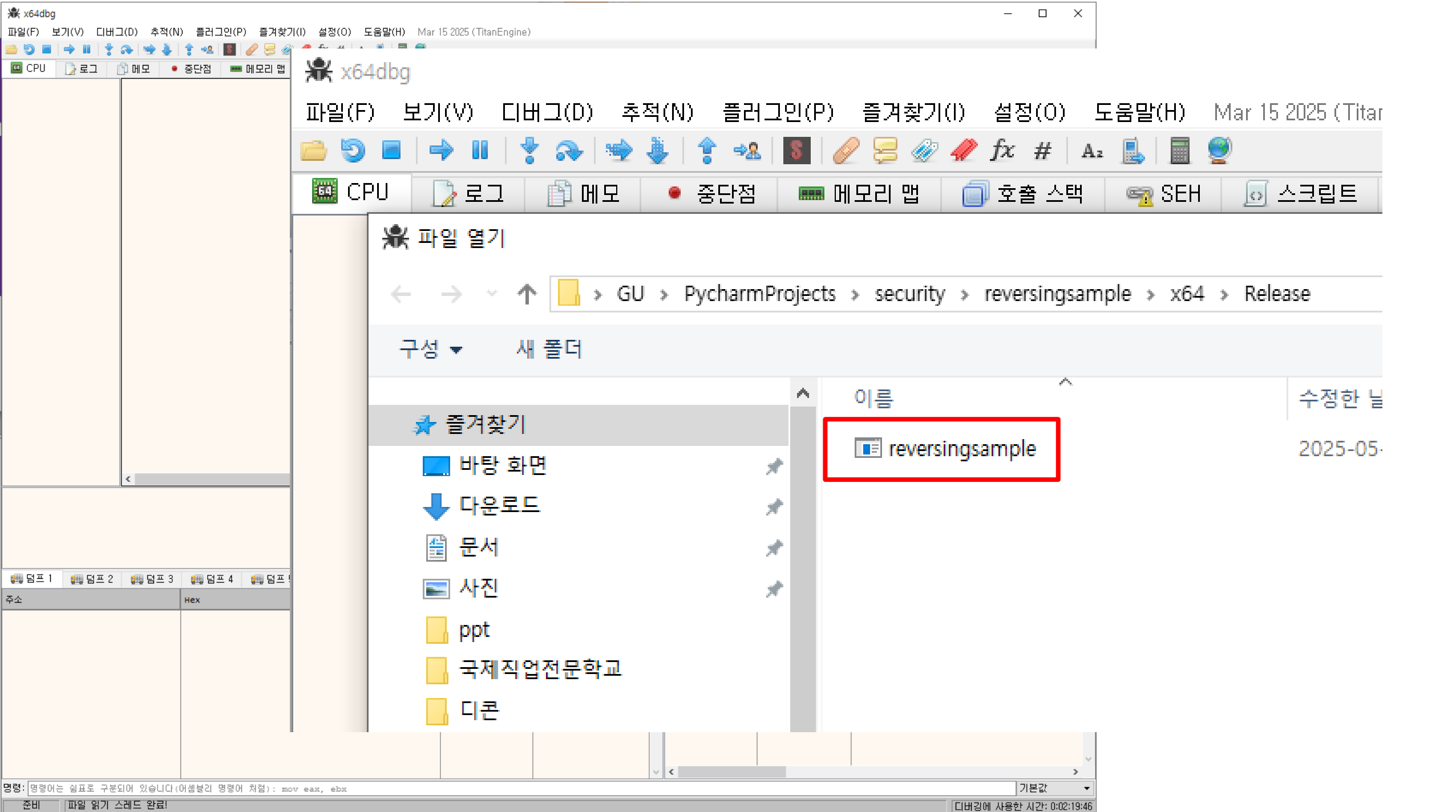
[Parent folder](#)

symbols-snapshot_2025-03-15_15-57.zip	2025-03-15	25.0 MB	409	
snapshot_2025-03-15_15-57.zip	2025-03-15	35.0 MB	12,128	
symbols-snapshot_2025-03-03_18-32.zip	2025-03-03	25.0 MB	5	
snapshot_2025-03-03_18-32.zip	2025-03-03	35.0 MB	20	

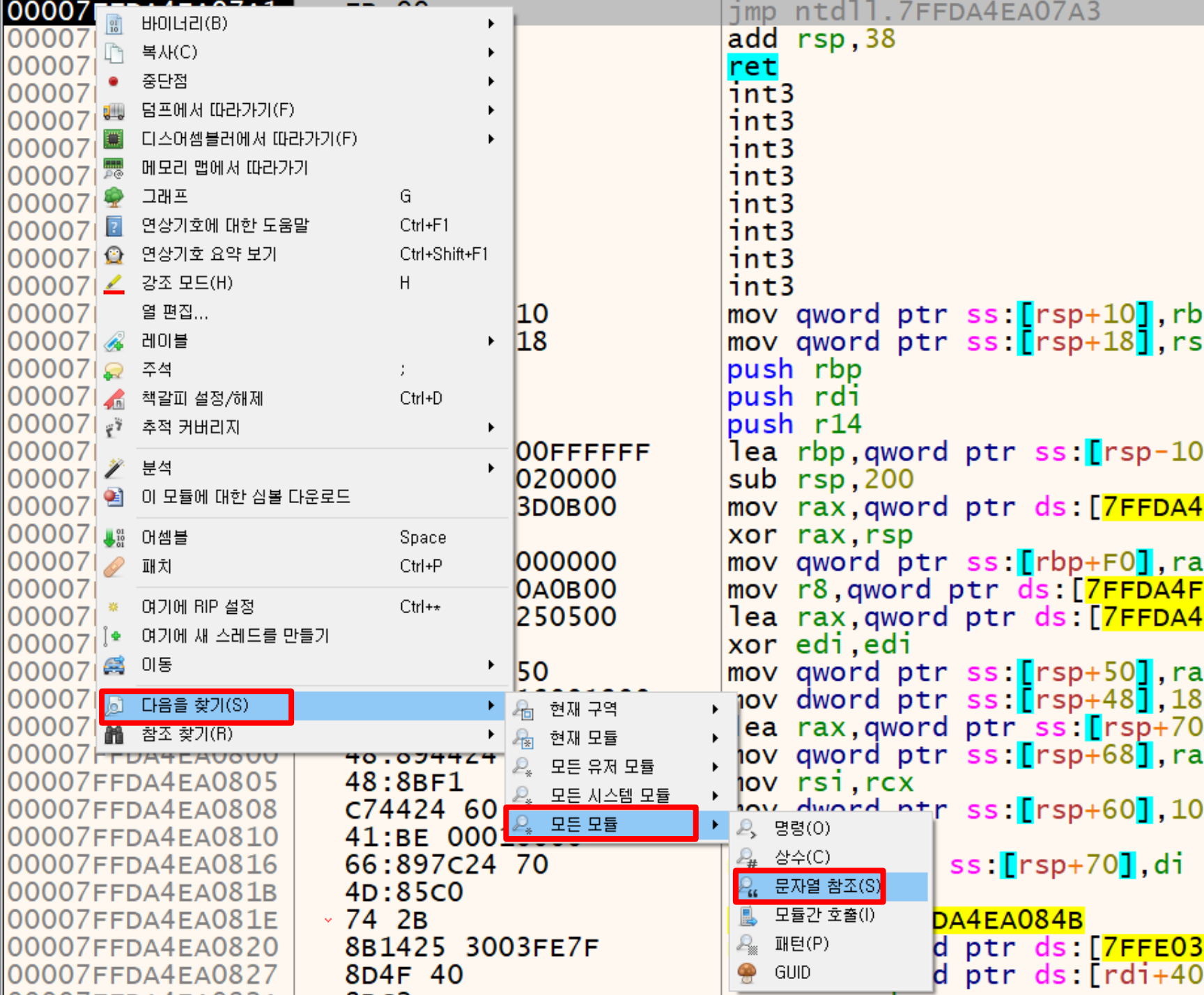
1. 압축풀기

2. C:\Users\GU\Downloads\snapshot_2025-03-15_15-57\release\x64

3. x64dbg.exe 실행



	00007FFDA4EA0790	E8 EBD1FCFF	call <ntdll.NtQueryInformationThread>	
	00007FFDA4EA0795	85C0	test eax,eax	
	00007FFDA4EA0797	< 78 0A	js ntdll.7FFDA4EA07A3	
	00007FFDA4EA0799	807C24 40 00	cmp byte ptr ss:[rsp+40],0	
	00007FFDA4EA079E	< 75 03	jne ntdll.7FFDA4EA07A3	
	00007FFDA4EA07A0	CC	int3	
RIP	00007FFDA4EA07A1	< EB 00	jmp ntdll.7FFDA4EA07A3	
	00007FFDA4EA07A3	< 48:83C4 38	add rsp,38	
	00007FFDA4EA07A7	C3	ret	
	00007FFDA4EA07A8	CC	int3	
	00007FFDA4EA07A9	CC	int3	
	00007FFDA4EA07AA	CC	int3	
	00007FFDA4EA07AB	CC	int3	
	00007FFDA4EA07AC	CC	int3	
	00007FFDA4EA07AD	CC	int3	
	00007FFDA4EA07AE	CC	int3	
	00007FFDA4EA07AF	CC	int3	
	00007FFDA4EA07B0	48:895C24 10	mov qword ptr ss:[rsp+10],rbx	
	00007FFDA4EA07B5	48:897424 18	mov qword ptr ss:[rsp+18],rsi	
	00007FFDA4EA07BA	55	push rbp	
	00007FFDA4EA07BB	57	push rdi	
	00007FFDA4EA07BC	41:56	push r14	
	00007FFDA4EA07BE	48:8DAC24 00FFFFFF	lea rbp,qword ptr ss:[rsp-100]	
	00007FFDA4EA07C6	48:81EC 00020000	sub rsp,200	
	00007FFDA4EA07CD	48:8B05 3C3D0B00	mov rax,qword ptr ds:[7FFDA4F54510]	
	00007FFDA4EA07D4	48:33C4	xor rax,rsp	
	00007FFDA4EA07D7	48:8985 F0000000	mov qword ptr ss:[rbp+F0],rax	
	00007FFDA4EA07DE	4C:8B05 230A0B00	mov r8,qword ptr ds:[7FFDA4F51208]	
	00007FFDA4EA07E5	48:8D05 DC250500	lea rax,qword ptr ds:[7FFDA4EF2DC8]	00007FFDA4EF2DC8:L
	00007FFDA4EA07EC	33FF	xor edi,edi	
	00007FFDA4EA07EE	48:894424 50	mov qword ptr ss:[rsp+50],rax	
	00007FFDA4EA07F3	C74424 48 16001800	mov dword ptr ss:[rsp+48],180016	
	00007FFDA4EA07FB	48:8D4424 70	lea rax,qword ptr ss:[rsp+70]	
	00007FFDA4EA0800	48:894424 68	mov qword ptr ss:[rsp+68],rax	
	00007FFDA4EA0805	48:8BF1	mov rsi,rcx	
	00007FFDA4EA0808	C74424 60 00000001	mov dword ptr ss:[rsp+60],1000000	



- 마우스오른쪽 클릭
- 다음을 찾기
- 모든 모듈
- 문자열 참조

문자열 주소	문자열
00007FF620D92260	"hello, world\n"
00007FFD9F3F4490	"bad exception"
00007FFD9F3F4460	"Unknown exception"
00007FFD9F3F44E8	"Access violation - r
00007FFD9F3F44E8	"Access violation - r
00007FFD9F3F4560	"Bad dynamic_cast!"
00007FFD9F3F4510	"Attempted a typeid c
00007FFD9F3F4538	"Bad read pointer - r
00007FFD9F3F44E8	"Access violation - r
00007FFD9F3F4628	L"api-ms-"
00007FFD9F3F4640	"FlsAlloc"
00007FFD9F3F4640	"FlsAlloc"
00007FFD9F3F4658	"FlsFree"

00007FF620D91070	40:53	push rbx
00007FF620D91072	48:83EC 20	sub rsp,20
00007FF620D91076	BB 040000	mov ebx,4
00007FF620D9107B	0F1F4400	nop dword ptr ds:[rax+rax],eax
00007FF620D91080	48:8D0D D8	lea rcx,qword ptr ds:["hello, worl
00007FF620D91087	E8 84FFFF	call <reversingsample.printf>
00007FF620D9108C	48:83EB 01	sub rbx,1
00007FF620D91090	75 EE	jne reversingsample.7FF620D91080
00007FF620D91092	FF15 E010	call qword ptr ds:[&getchar]
00007FF620D91098	33C0	xor eax,eax
00007FF620D9109A	48:83C4 20	add rsp,20
00007FF620D9109E	5B	pop rbx
00007FF620D9109F	C3	ret
00007FF620D910A0	CC	int3
00007FF620D910A1	CC	int3
00007FF620D910A2	CC	int3
00007FF620D910A3	CC	int3
00007FF620D910A4	CC	int3
00007FF620D910A5	CC	int3
00007FF620D910A6	6666:0F1F	nop word ptr ds:[rax+rax],ax
00007FF620D910B0	48:3B0D 48	cmp rcx,qword ptr ds:[<__security_c
00007FF620D910B7	75 10	jne <reversingsample.ReportFailure>
00007FF620D910B9	48:C1C1 10	rol rcx,10
00007FF620D910BD	66:F7C1 F1	test cx,FFFF
00007FF620D910C2	75 01	jne <reversingsample.RestoreRcx>
00007FF620D910C4	C3	ret
00007FF620D910C5	48:C1C9 10	ror rcx,10
00007FF620D910C9	E9 AA0200	jmp <reversingsample.__report_gsfa
00007FF620D910CE	CC	int3


```
push rbx
sub rsp,20
mov ebx,4
nop dword ptr ds:[rax+rax],eax
lea rcx,qword ptr ds:["hello, world\n"...>]
call <reversingsample.printf>
sub rbx,1
jne reversingsample.7FF620D91080
call qword ptr ds:["&getchar"]
xor eax,eax
add rsp,20
pop rbx
```

rbx = 4



"hello, world" 출력



rbx = rbx - 1



rbx != 0 → 다시 출력 반복



getchar() → 입력 대기



return 0

```
import dis
```

```
def my_function():
```

```
    i = 0
```

```
    while i < 4:
```

```
        print("Hello World\n")
```

```
        i = i + 1
```

```
dis.dis(my_function)
```

파이썬 바이트코드

바이트코드(Bytecode)는 고급 언어로 작성된 소스 코드를 가상머신이 이해할 수 있는 중간 코드로 컴파일
가상머신을 위한 기계어

```

4          0 LOAD_CONST          1 (0)
          2 STORE_FAST          0 (i)

5  >>     4 LOAD_FAST            0 (i)
          6 LOAD_CONST          2 (4)
          8 COMPARE_OP          0 (<)
         10 POP_JUMP_IF_FALSE    30

6          12 LOAD_GLOBAL        0 (print)
          14 LOAD_CONST          3 ('Hello world\n')
          16 CALL_FUNCTION        1
          18 POP_TOP

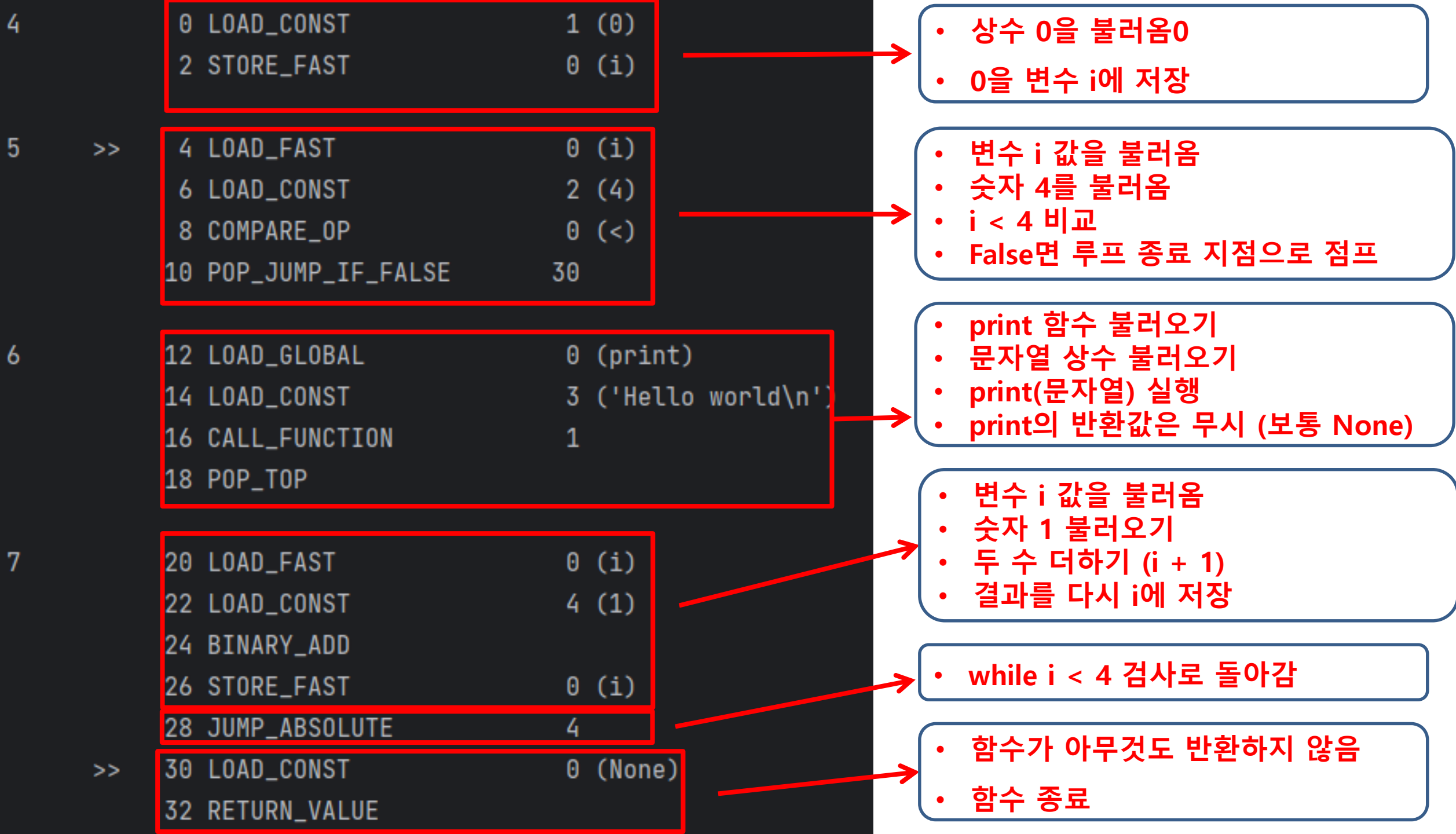
7          20 LOAD_FAST            0 (i)
          22 LOAD_CONST          4 (1)
          24 BINARY_ADD
          26 STORE_FAST          0 (i)
          28 JUMP_ABSOLUTE       4
>>     30 LOAD_CONST          0 (None)
         32 RETURN_VALUE

```

파이썬 인터프리터

코드를 어떻게 한 줄씩 실행할지
저수준 명령어로 변환한 모습

1. $i = 0$
2. $i < 4 \rightarrow \text{True} \rightarrow$ 루프 시작
3. "Hello world" 출력
4. $i = i + 1$
5. 다시 2로 반복
6. $i == 4 \rightarrow \text{False} \rightarrow$ 함수 종료



Python » **Korean | 한국어**

3.13.3

3.13.3 Documentation » 파이썬 표준 라이브러리 » 파이썬 언어 서비스 » **dis** — Disassembler for Python bytecode

Python bytecode

Theme

[전체](#) [이미지](#) [쇼핑](#)

Python docume

<https://docs.python.org/3.13.3/library/dis.html>

dis — Disassembler for Python bytecode

The `dis` module supports disassembling Python bytecode which this module provides.

목차

`dis` — Disassembler for Python bytecode

- Command-line interface
- 바이트 코드 분석
- 분석 함수
- 파이썬 바이트 코드 명령어
- 옴코드 모음

이전 항목

`compileall` — 파이썬 라이브러리 바이트 컴파일하기

다음 항목

`pickletools` — 피클 개발자를 위한 도구

이 페이지

[버그 보고하기](#)
[소스 보기](#)

dis — Disassembler for Python bytecode

소스 코드: [Lib/dis.py](#)

`dis` 모듈은 CPython [바이트 코드](#)를 역 어셈블 하여 분석을 지원합니다. 이 모듈이 입력으로 취하는 파일 `Include/opcode.h`에 정의되어 있으며 컴파일러와 인터프리터에서 사용됩니다.

바이트 코드는 CPython 인터프리터의 구현 세부 사항입니다. 파이썬 버전 간에 바이트 코드가 추후 변경될 것이라는 보장은 없습니다. 이 모듈을 사용하는 것이 파이썬 VM이나 파이썬 릴리스에 걸쳐 작동해야 합니다.

버전 3.6에서 변경: 명령어마다 2바이트를 사용합니다. 이전에는 바이트 수가 명령어에 따라 달랐습니다.

버전 3.10에서 변경: The argument of jump, exception handling and loop instructions is now a relative offset rather than the byte offset.

버전 3.11에서 변경: Some instructions are accompanied by one or more inline cache entries in the form of [CACHE](#) instructions. These instructions are hidden by default, but can be shown by passing `show_caches=True` to any `dis` utility. Furthermore, the interpreter now adapts the bytecode for different runtime conditions. The adaptive bytecode can be shown by passing `adaptive=True` to any `dis` utility.

버전 3.12에서 변경: The argument of a jump is the offset of the target instruction relative to the instruction following the jump.

1. 바이트 코드 변환

- `py_compile`

2. 바이트코드 -> 파이썬 파일 복원

- `pip install uncompyle6`

3. Exe 파일 제작

4. Exe -> pyc 추출

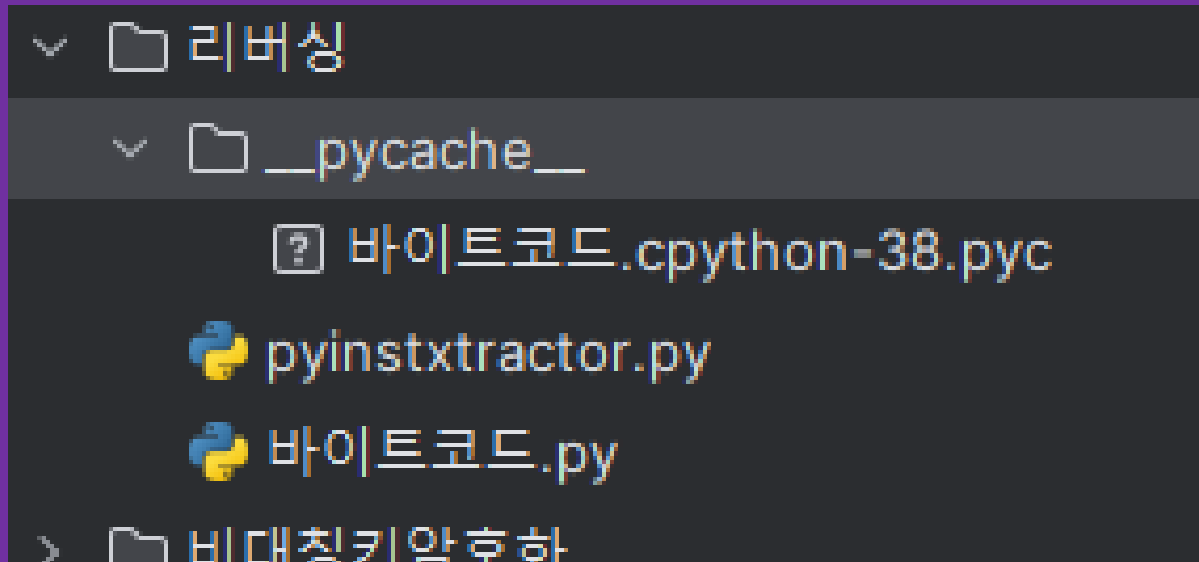
5. Pyc -> py 추출

- `uncompyle6`

실습

1. 바이트 코드 변환

`python -m py_compile 바이트코드.py`

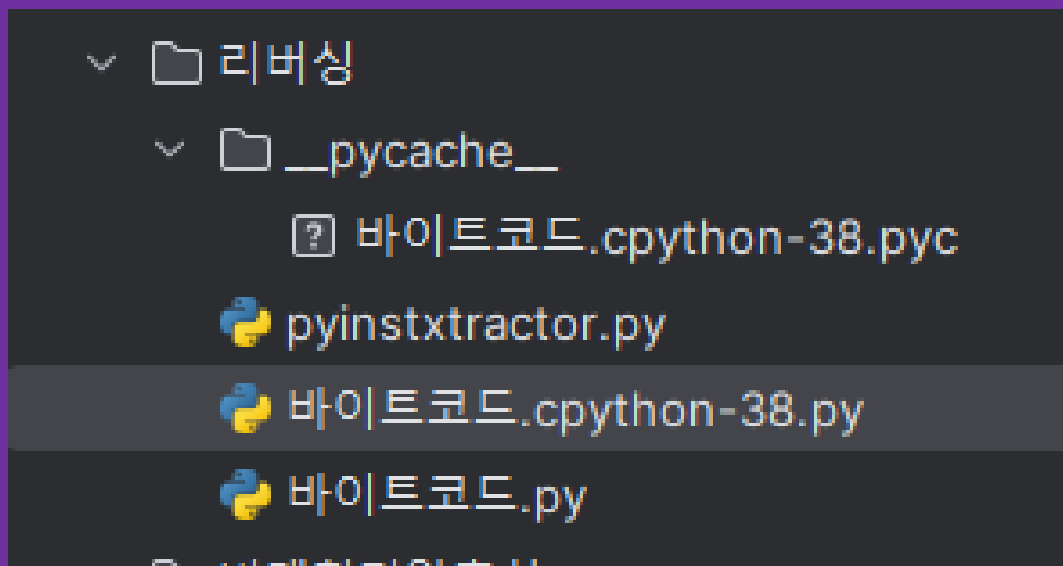


`__pycache__` 폴더안에
`바이트코드.cpython-38.pyc` 생성

실습

1. Pyc -> py 파일 복원

`uncompyle6 -o . ./__pycache__/바이트코드.cpython-38.pyc`



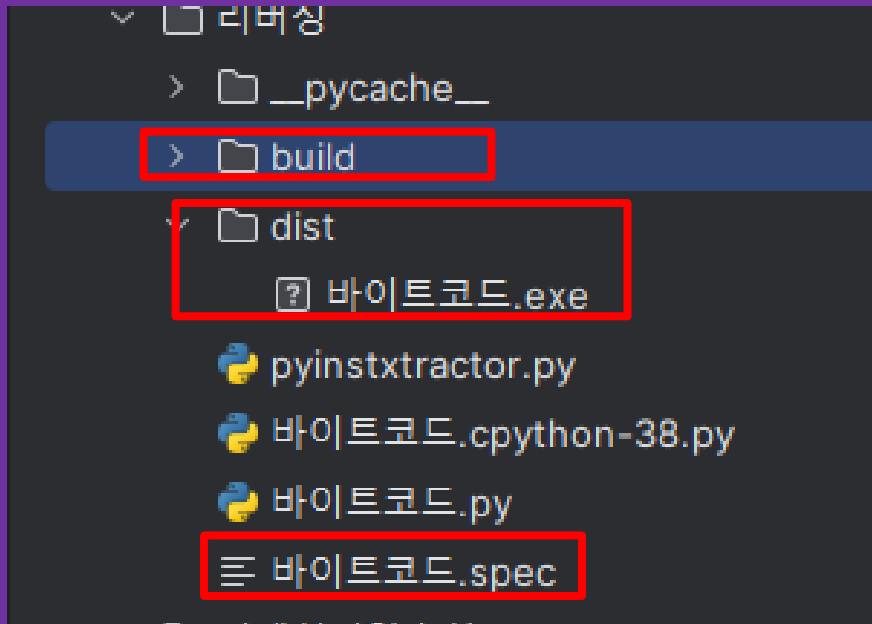
바이트코드.cpython-38.py 생성

pip install uncompyle6 설치

실습

1. EXE 파일 생성

`pyinstaller --onefile .\#바이트코드.py`

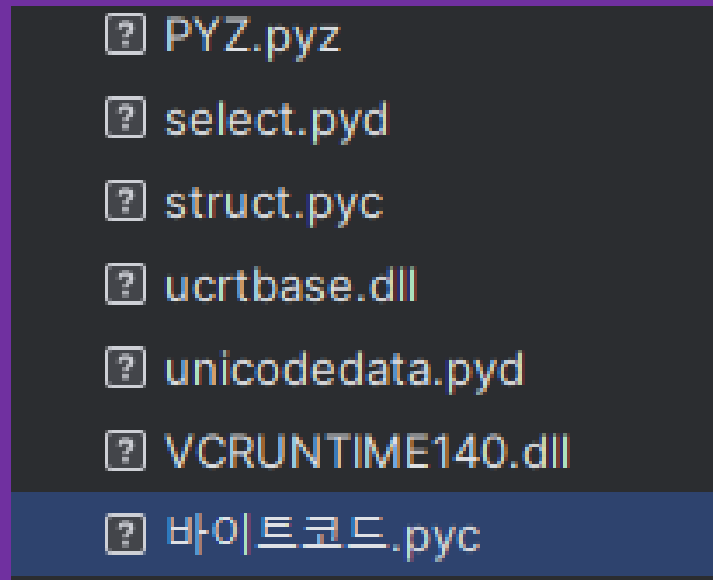
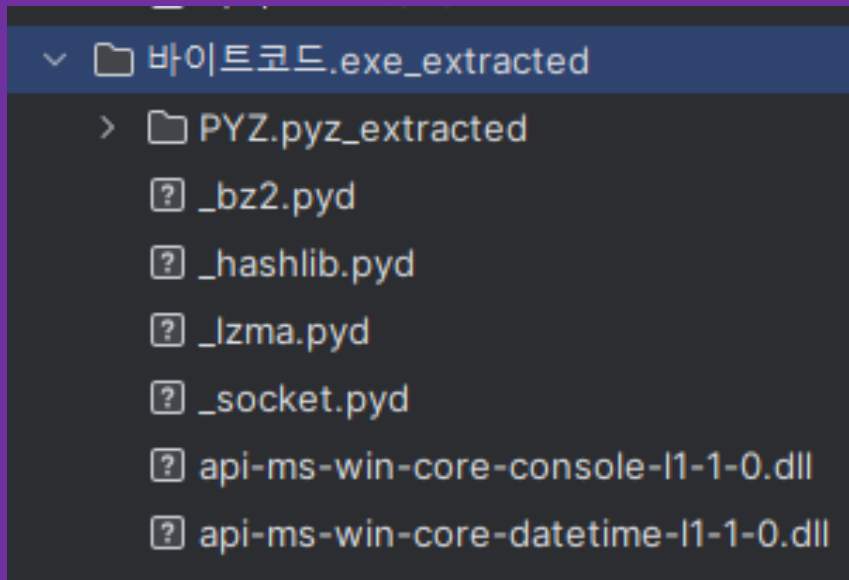


pip install pyinstaller 설치

실습

1. EXE -> pyc(바이트코드) 파일 추출

`pyinstxtractor.py .\dist\바이트코드.exe`



바이트코드.pyc 생성

pyinstxtractor.py 다운로드

실습

1. pyc(바이트코드) -> py 파일 복원

`uncompyle6 .\#바이트코드.exe_extracted\#바이트코드.pyc`

```
(venv) PS C:\Users\GU\PycharmProjects\security\리버싱> uncompyle6 .\바이트코드.exe_extracted\바이트코드.pyc
```

```
# uncompyle6 version 3.9.2
```

```
# Python bytecode version base 3.8.0 (3413)
```

```
import dis
```

```
def my_function():
```

```
    i = 0
```

```
# Decompiled from: Python 3.8.10 (tags/v3.8.10:3d8993a, May 3 2021, 11:48:03) [MSC v.1928 64 bit (AMD64)]
```

```
# Embedded file name: 바이트코드.py
```

```
import dis
```

```
def my_function():
```

```
    i = 0
```

```
    while i < 4:
```

```
        print("Hello World\n")
```

```
        i = i + 1
```

```
dis.dis(my_function)
```

바이트코드.cpython-38.py